Representing Data and Records

CS157B

Chris Pollett

Feb 7, 2005.

Outline

- Representing Relational Data
- Representing Objects
- Representing XML
- Fixed Length Records
- Record Headers
- Packing

Representing Data in Secondary Storage

How do we store relations and objects to Disk?

- Attributes are stored as fixed or variable length sequences of bytes called ``fields''
- Fields in turn are organized into ``records'' which correspond to tuples or objects.
- Records need to be stored into disk blocks
- Finally, all the blocks used to store a relation or object are stored as a *file*. This might have additional structures associated with it such as indices.

Example

Consider:

```
CREATE TABLE MovieStar(
    name CHAR(30) PRIMARY KEY,
    address VARCHAR(255),
    gender CHAR(1),
    birthdate DATE
);
```

- Need to explain: how to convert CHAR, VARCHAR, etc to fields; how to put the whole tuple into a record and then blocks, files, etc.
- Need to explain: how to handle different sized records; what happens when fields change in size

OO Extensions

An object is similar to a tuple. Its field act like attributes. There are two important extensions, however:

1. Objects can have methods
2. Objects may have an object identifier (OID), which is an address that refers uniquely to that object. Moreover, objects can have relationships to other objects which are represented as pointers.

Methods are usually stored with the schema. To access these an object record needs to say what class it belongs to.

XML Extensions

- Recall XML is like HTML but can define own tags. Example: <location><street>Rue des Examples</street><city>Grande Ville</city><state>CA</state></location>
- The allowed tags for a document specified in a DTD or in an XML Schema.
- Documents have a tree-like structure. When stored can separate structure from data in separate blocks or have all together. The tree-like structure can also be compressed or not. There are also issues about labelling nodes for easy update.

Representing Data Elements I

We now discuss how to represent various kinds of data elements:

- **Fixed length strings** - i.e., CHAR(n) fields for some fixed n. To store these we just use an array of size n. If a value has fewer than n bytes we use a special pad character.

  For example, if we have type CHAR(5), a possible value `cat' might be stored as: cat**

Representing Data Elements II

- **Variable-Length Character Strings** - Suppose we had an attribute of type VARCHAR(n). Here n must be less than 255. We could represent this in several different ways:
  - **Length plus content**. Allocate n+1 bytes. Use the first byte for the length and the remaining bytes for the content. If left over bytes ignore.
  - **Null-terminated string.** Allocate n+1 bytes. Write string followed by a \0. Anything after the \0 is ignored.

Representing Data Elements III

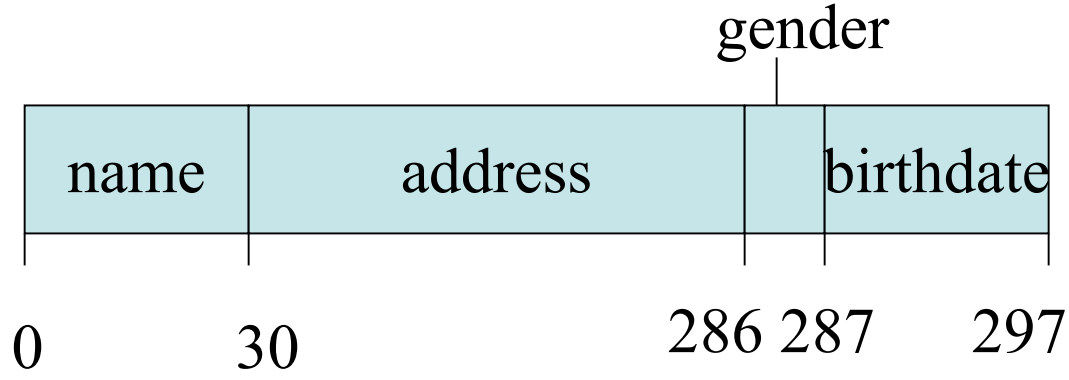
- Date and Time - These are usually stored as a fixed length character string following some format. For example, might use format YYYY-MM-DD and have a value like ‘2005-02-07’. Similarly, a time might have a format like HH:MM:SS.FF (SQL2) and have a value like ‘23:05:02.35’. The .35 is the fraction of a second.

Representing Data Elements IV

- Bits - The SQL2 type BIT(n) is used when one wants sequences of n bits. To store this we pack the bits into [n/8] (round up) consecutive bytes. Unused bits in last byte are ignored. For example, suppose we want to store a BIT(6) value 001100. We might represent this with 00110000. Note sometimes store a single bit as 00000000 or 11111111.
- Enumerated Types - {RED, GREEN, YELLOW}. These can be represented using numbers 0, 1, 2 and store in an integer.

Fixed-Length Records

- Let's look at how a MovieStar tuple from our earlier CREATE TABLE would be stored:



- Note this may be inefficient to process by 32 or 64 bit processors. Sometimes reorganize records so that each field divisible by 4 or 8 to help the processor.

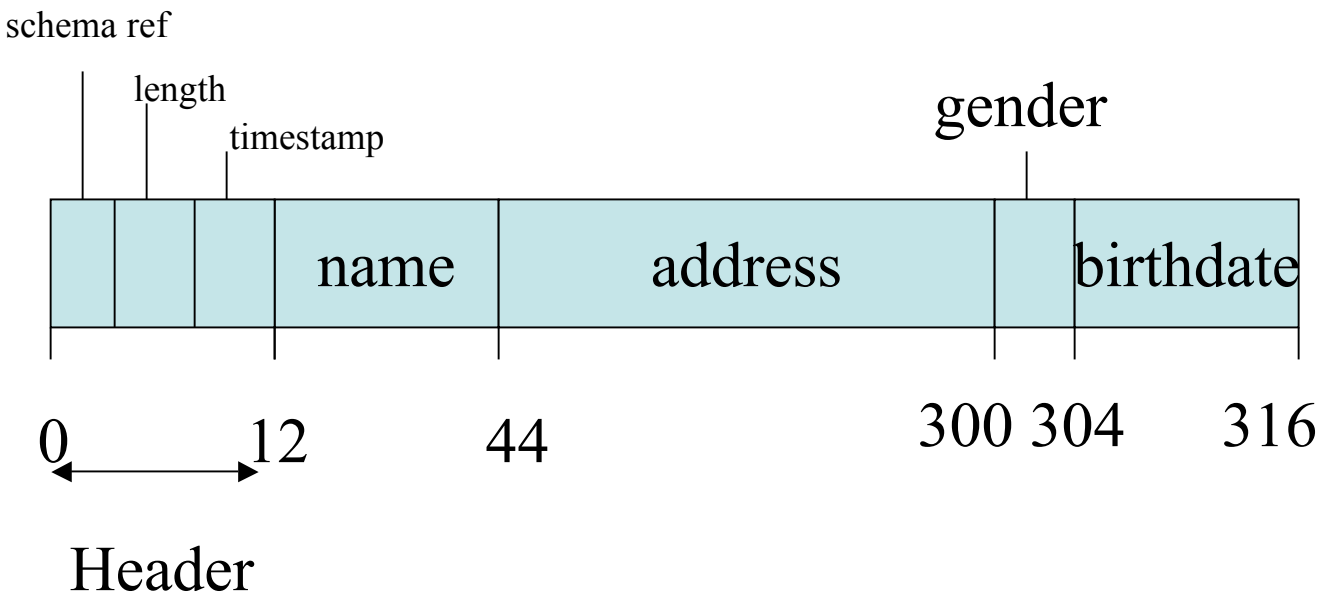
Record Headers

- Sometimes need to store addition info besides the value of each field. This data could be things like:
  1. The record schema or a pointer to it.
  2. The length of the record
  3. Timestamp indicating when the data was last modified or read.

Schema Info

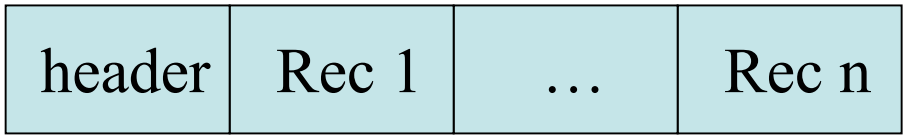
- When a CREATE TABLE is done the database stores:
  - the attributes of the relation
  - their types
  - the order in which the attributes appear in the tuple
  - any constraints on the attributes.
- This info is usually store in the catalog/dictionary of the database. Each record then for a given relation keeps a pointer to where in the dictionary the information is stored.

Example with Header and Optimized for 32 bit processing



Packing Fixed-Length Records into Blocks

- A block might look like:



- The header above is called a block header and might store: links to other blocks, role played by this block in network of blocks, info about which relation tuples in this block belong to, a directory giving offsets to records in this block, a block ID, and a timestamp for when this block was last read/modified.